A Parallel Computational Database Pattern

Information exchange during setup

A Parallel Computational Database Pattern

> 100M objects

> terabyte size data

Select objects with particular property value

A Parallel Computational Database Pattern

Time, space $\sim$ model size / nhost

BG/Q: 4 racks, 1024 nodes/rack

16 GB/node $\longrightarrow$ 64 TB

32 ranks/node $\longrightarrow$ 0.5 GB

A Parallel Computational Database Pattern

Data is read, generated, analyzed to instantiate the model

Communication should not unreasonably dominate

Hard disk read/write is slow. Avoid as much as possible.

A Parallel Computational Database Pattern

Ranks have data, ranks want data,
No one knows who has or wants.

{key : data}
key is integer, float, string, tuple.

All ranks participate in alternating computation, communication.

Example: MPI_ISend/Recv spike exchange

Cells do not know which ranks are interested in its spikes.

Example: MPI_ISend/Recv spike exchange

Cells do not know which ranks are interested in its spikes.

Example: Source/Target connectivity

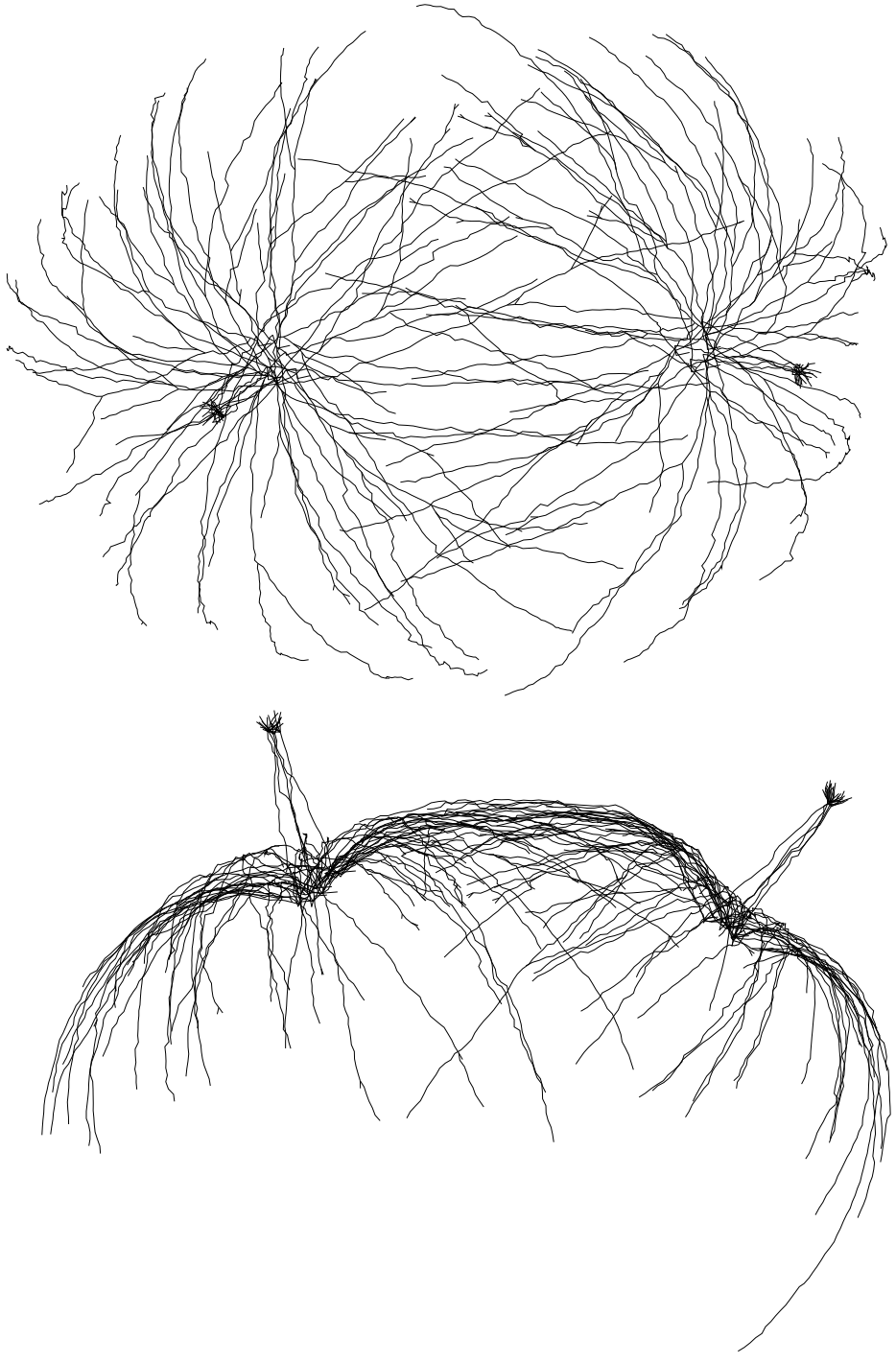
Reciprocal synapse connection description.

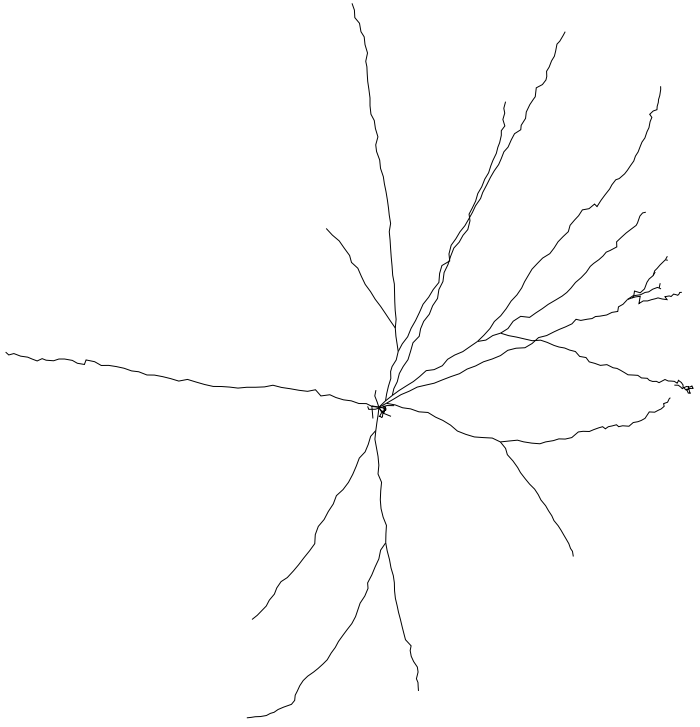
(mitral_gid, mdend_index, xm, granule_gid, gdend_index, xg, ...)

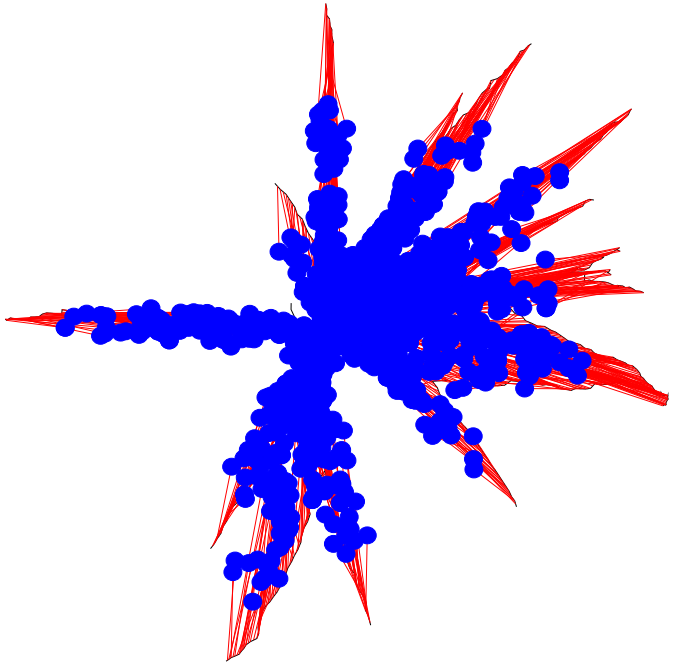
Construct a mitral => all the tuples with that mitral_gid.

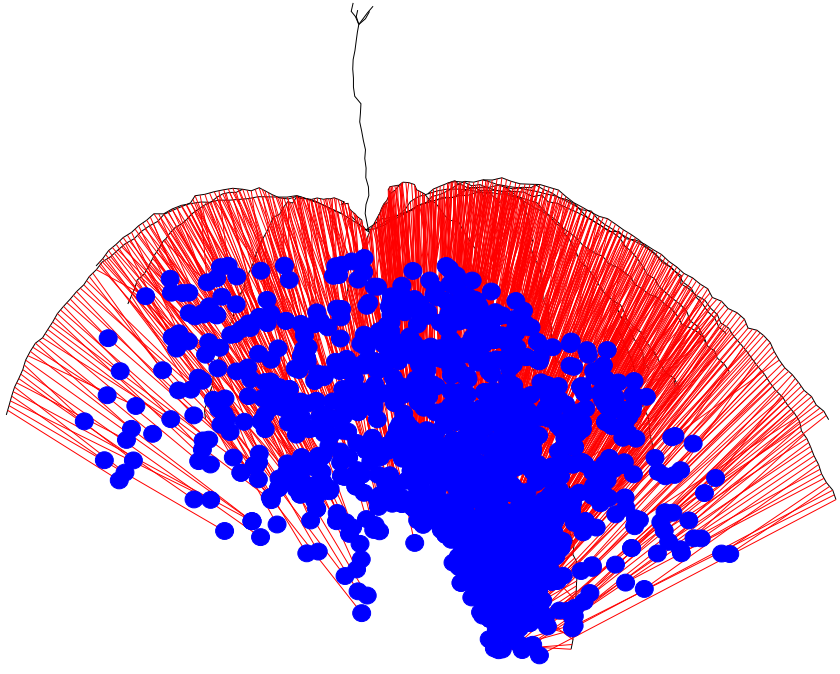
Granules don’t know enough for construction of the tuples.

Construct a granule => gather all the tuples with that granule_gid.









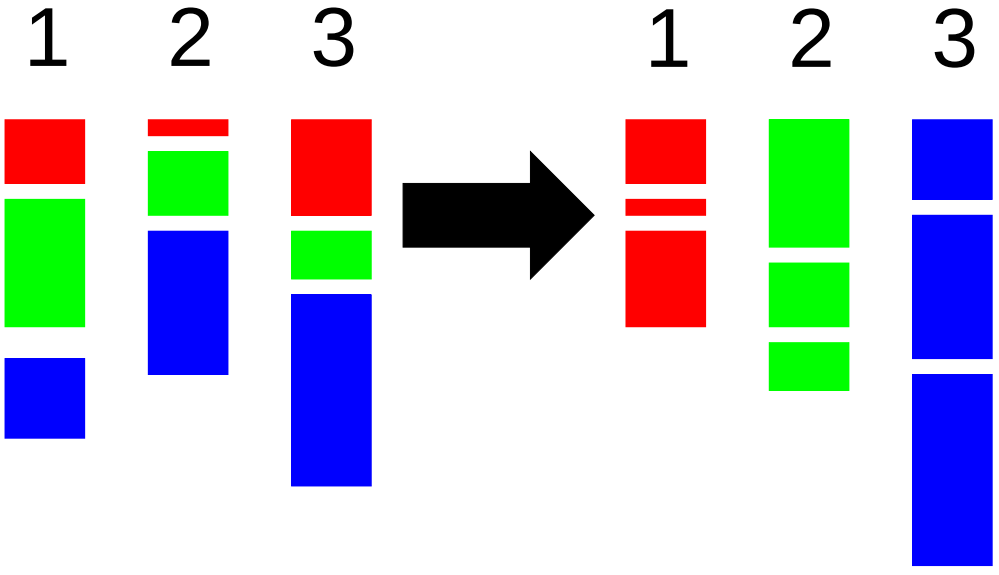
Basic exchange:

dest = ParallelContext.py_alltoall(src)

src and dest are a list of nhost pickleable objects.

src[j] on the ith rank will be copied to dest[i] on the jth rank.

Identical to dest = mpi4py.MPI.WORLD.alltoall(src).



Basic exchange:

dest = ParallelContext.py_alltoall(src)

src and dest are a list of nhost pickleable objects.

src[j] on the ith rank will be copied to dest[i] on the jth rank.

Identical to dest = mpi4py.MPI.WORLD.alltoall(src).

Essentially a wrapper for:

```
MPI_Alltoallv(s, scnt, sdispl, MPI_CHAR,
                        r, rcnt, rdispl, MPI_CHAR, comm);
```

along with a preliminary
MPI_all2all(scnt, 1, MPI_INT, rcnt, 1, MPI_INT, comm);
in order to calculate rcnt and rdispl.

But:

No one knows who holds what.

No room for anyone to have a global map.

But:

No one knows who holds what.

No room for anyone to have a global map.

Solution: A rendezvous rank function:

rank = rendezvous(property)

usually

rank = gid % nhost

But:

No one knows who holds what.

No room for anyone to have a global map.

Solution: A rendezvous rank function:

rank = rendezvous(property)

less often

```
def key2rank(property):
    return property.__str__().__hash__() % nhost
```

But:

No one knows who holds what.

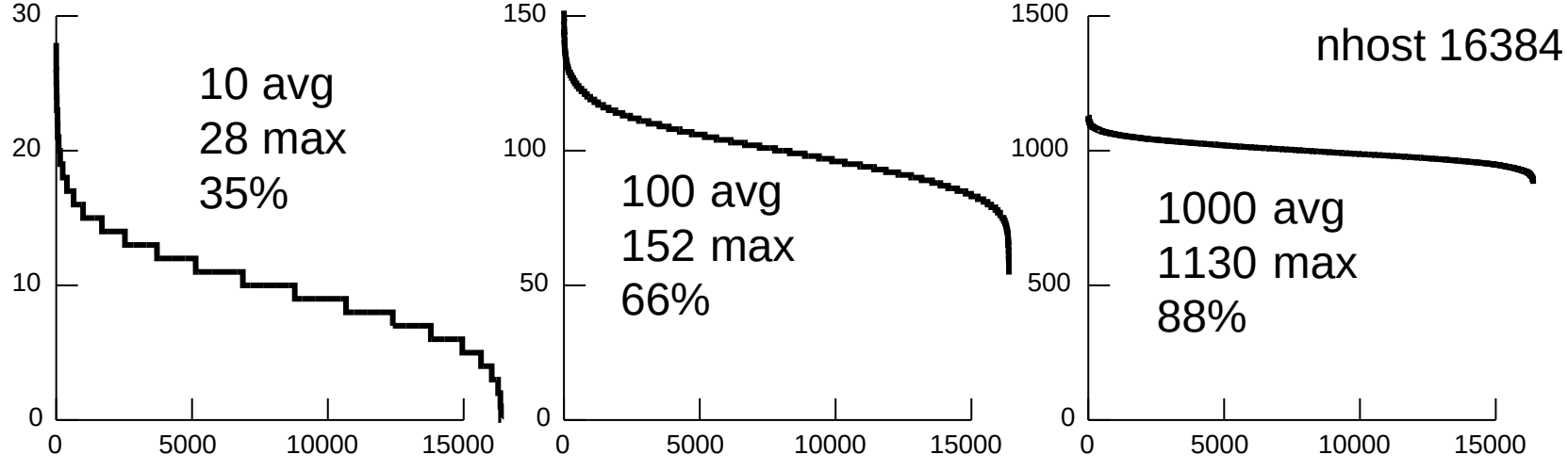
No room for anyone to have a global map.

Solution: A rendezvous rank function:

```
rank = rendezvous(property)
```

less often

```
def key2rank(property):
    return property.__str__().__hash__() % nhost
```



But:

No one knows who holds what.

No room for anyone to have a global map.

Solution: A rendezvous rank function:

rank = rendezvous(property)

usually

rank = gid % nhost

1) Everyone sends the keys they own to the rendezvous rank.
2) Everyone sends the keys they want to the rendezvous rank.
3) The rendezvous rank sends back to the owners, which ranks want which keys.
4) The owners send the objects to the ranks that want them.

Usually simplification is possible:

If the objects are small.

1) Everyone sends the keys and objects they own to the rendezvous rank.
2) Everyone sends the keys they want to the rendezvous rank.
3) The rendezvous rank sends the objects to the ranks that want them.

Usually simplification is possible:

If rendezvous(property) is known to be the source rank for all the keys (a−priori or by verifying with an all_reduce).

1) Everyone sends the keys they want to the owner ranks.

2) The owners send the objects to the ranks that want them.

Usually simplification is possible:

If rendezvous(property) is known to be the destination rank for all the keys (a−priori or by verifying with an all_reduce).

1) The owners send the objects to the ranks that want them.